Question 1

The answer is **D**. Straight from the definition of big O.

Question 2

- h[1] = 13
- h[2] = 4
- h[5] = 15
- h[6] = 23
- h[11] = 22
- h[12] = 19
- h[24] = 71
- h[25] = 81

Jacob:

1. 16
2. 12
3. 18
4. 25
5. 41
6. 30
7. 99
8. 54

Questions 3-14

These came from a bank and were randomly ordered. Here they are:

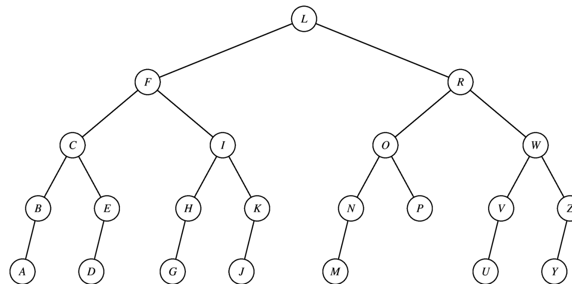
- *O(1)*. Sets don't insert duplicates, so the set size is always 1.
- $O(n^2)$
- *O(log n)*
- *O(n)* -- that's how long it takes to traverse the multiset. I also gave credit to *O(m log n)*, which you can achieve using *upper_bound*. We'll learn more about that next semester.
- *O(n)* - Tree traversal.
- *O(log n)*
- *O(n) - That's what's special about a heap.*
- *O(log n)*
- *O(log n)*
- *O(n log n)*
- *O(n). It's O(1) to get to the 3rd element, and O(1) for each insertion.*
- *O(n). push_front() on a deque is O(1).*
-

Question 15

In the following blank, please enter a preorder printing of the nodes. You can just enter all of the letters, in order, without spaces: **TBHDENLSUW**

And in the following blank, please enter a postorder printing of the nodes: **EDLSNHBWUT**

Question 16



Questions 17 through 20

Answers: J, 2.

Answers: P, 1.

Answers: Y, 1.

Answers: U, 2.

Question 21

The destructor needs to typecast **s** to an instance of **MyZippy ***. Then it should delete any integers allocated in the vector. Finally, it should delete the instance of **MyZippy ***. You don't need to clear the vector, because that is done automatically in the last delete call:

```
Zippy::~Zippy()
{
  size_t i;
  MyZippy *z;

  z = (MyZippy *) s;
  for (i = 0; i < z->v.size(); i++) delete z->v[i];
  delete z;
}
```

Question 22

```
Dnode *Dlist::Begin()
{
  return sentinel->flink;
}

Dnode *Dlist::End()
{
  return sentinel;
}

void Dlist::Insert_Before(const string &s, Dnode *n)
{
  Dnode *newnode, *prev;

  newnode = new Dnode;
  prev = n->blink;

  newnode->s = s;
  newnode->flink = n;
  newnode->blink = prev;   // this could be newnode->blink = n->blink;
  n->blink = newnode;
  prev->flink = newnode;   // this could be newnode->blink->flink = newnode;
  size++;
}
```

Question 23

```
double rank(const Treenode *n)
{
  double total, average;
  size_t i;

  /* Base case is when a node has no children. */

  if (n->children.size() == 0) return weight;

  /* Otherwise, compute the average rank of the children. */
  total = 0;
  for (i = 0; i < n->children.size(); i++) {
    total += rank(n->children[i]);
  }
  average = total / (double) n->children.size();

  /* Return the maximum */
  return (weight > average) ? weight : average;
}
```

Question 24

```
11 12 10
68 88 19 97 84 91 50
```